# Styles

In dit onderdeel gaan we het hebben over Styles in Xamarin Forms. Hierin gaan we het bijvoorbeeld hebben over:

* *Global Styles*
* *Style Inheritence*
* *Dynamic Styles*

Xamarin Forms applicaties hebben meestal meerdere controls, die controls hebben dan allemaal een identiek uiterlijk. Het instellen van zo’n uiterlijk kan heel vervelend zijn en veel werk in beslag nemen. Daarom is er Xamarin Forms Styles functionaliteit, met deze functionaliteit word het makkelijker gemaakt om stijlen toe te passen in het design van je applicatie door het toepassen van grouping en settings properties aan passen.

# Global Styles:

## Het creëren van een global style in Xamarin met XAML

Over het algemeen zijn alle Xamarin.Forms applicaties gemaakt met behulp van het implementeren van een ‘Application’ subclass. Om een style te kunnen toe te kunnen wijzen in de ‘ResourceDictionary’ met XAML, moet de standaad App classen vervangen worden door een XAML App Class en de code erachter.

Dit stuk code laat zien hoe dit is gedaan in C#:

public App ()

{

var buttonStyle = new Style (typeof(Button)) {

Setters = {

...

new Setter { Property = Button.TextColorProperty, Value = Color.Teal }

}

};

Resources = new ResourceDictionary ();

Resources.Add ("buttonStyle", buttonStyle);

...

}

Button Style property ResourceDictionary en het toevoegen van de button style aan de Dictionary

# Style Inheritence

Style inheritence is het toepassen van stijlen die al bestaan met behulp van de Style.BasedOn Stijlen die erfen van andere base stijlen kan Setters instances bevatten voor nieuwe properties, of het gebruiken om het te overschrijven van base stijlen. Stijlen die geërfd worden van een basis stijl moeten het zelfde typen hebben. Bijvoorbeeld als een basis style target “View” is, elke stijl die daar van word geërfd heeft automatisch de de TargetType=”View”.

De volgende code laat zien hoe je dit doet in XAML:

<Style x:Key="labelStyle" TargetType="Label"

BasedOn="{StaticResource baseStyle}">

...

<Setter Property="TextColor" Value="Teal" />

</Style>

<ContentPage.Content>

<StackLayout Padding="0,20,0,0">

<Label Text="These labels"

Style="{StaticResource labelStyle}" />

...

<Button Text="So is the button"

Style="{StaticResource buttonStyle}" />

</StackLayout>

</ContentPage.Content>

In C# geef je in plaats van een TargetType een typeof() mee, zie hier een voorbeeld

var labelStyle = new Style (typeof(Label)) {

BasedOn = baseStyle,

Setters = {

...

new Setter { Property = Label.TextColorProperty, Value = Color.Teal }

}

Children = {

new Label { Text = "These labels", Style = labelStyle },

...

new Button { Text = "So is the button", Style = buttonStyle }

}

De baseStyle die een View instance hebben de properties HorizontalOptions en VerticalOptions. De baseStyle is niet direct gekoppeld aan een control. Maar labelStyles en Button Styles erfen van de baseStyle.

# Dynamic Styles

De *DynamicResource* extensie is bijna hetzelfde als *StaticResource* want ze gebruiken allebei een *dictionary key* om de values op te vragen voor de stijl. Het enige verschil is dat bij een *StaticResource* maar een keer een in de dictionary kijkt of er aanpassing zijn gemaakt van bijvoorbeeld de values. Bij *DynamicResources* is dat anders het geval, namelijk dat de connectie met dictionary blijft bestaan en zo kan de app dus live zijn style aanpassen als deze value is aangepast.

De volgende code laat zien hoe de Dynamic Stijlen werkt in XAML

<ResourceDictionary>r

<Style x:Key="baseStyle" TargetType="View">

...

</Style>

<Style x:Key="blueSearchBarStyle"

TargetType="SearchBar"

BasedOn="{StaticResource baseStyle}">

...

<ContentPage.Content>

<StackLayout Padding="0,20,0,0">

<SearchBar Placeholder="These SearchBar controls"

Style="{DynamicResource searchBarStyle}" />

...

</StackLayout>

</ContentPage.Content>

Bijvoorbeeld als de OnButtonClicked event word aangeroepen, zal de searchBarStyle wisselen tussen de blueSearchBarStyle en greenSearchBarStyle. Dit zal als resultaat een andere kleur geven aan het uiterlijk van de app.

public DynamicStylesPage ()

{

InitializeComponent ();

Resources ["searchBarStyle"] = Resources ["blueSearchBarStyle"];

}

void OnButtonClicked (object sender, EventArgs e)

{

if (originalStyle) {

Resources ["searchBarStyle"] = Resources ["greenSearchBarStyle"];

originalStyle = false;

} else {

Resources ["searchBarStyle"] = Resources ["blueSearchBarStyle"];

originalStyle = true;

}

}

# Samenvatting

## Global Styles:

Stijlen kunnen global beschikbaar worden gesteld door hen toe te voegen aan de application’s *ResourceDictionary* Dit helpt om het dupliceren tegen te gaan van stijlen tussen pagina’s of controls op je app.

## Style Inheritence:

Stijlen kunnen geërfd worden van andere stijlen om tegen te gaan dat er teveel code staat wat eigenlijk niet nodig is. En zo werkt je applicatie dus sneller en heeft hij minder geheugen nodig. Stijl erving kan gedaan worden door het instellen van de *Style.BasedOn* property naar een bestaande Stijl.

## Dynamic Styles:

Standaard stijlen worden niet automatisch aangepast en blijven onaangepast zolang de app draait. Maar met de functie *Dynamic Styles* kan dat er dus voor zorgen dat je uiterlijk van je app live word aangepast als er een value word aangepast.

# Resourcedictionary

ResourceDictionary is een object of manager om alle resources bij te houden en herbruiken, aanpassen, toevoegen of verwijderen.

Je kan styles, control templates, data templates, colors en converters opslaan.

In XAML defineer je resources met StaticResource markup extensie. In c# defineer je met string-based index, maar het is makkelijker om direct om resources gelijk aan visueel element te koppelen zonder uit ResourceDictionary te halen.

Op welke niveau je defineer is belangrijk,

* Als je op het control defineer kan het control en zijn inheritted controls bij komen bij de resource
* Als je op het scherm defineer kan alle controls op het scherm bij de resource
* Als je op het app defineer kan alle schermen bij het resource.



Je roept het aan via x:Key=”naam van resources”

Als je op een lager niveau het zelfde resource roep neemt he laagste de hoogste over, zodat je specifiek aanpassing kan.

# Resources

Met resources kunt u data opslaan die u later in de programma kunt hergebruiken. U kunt een resource hergebruiken met twee types:

* Static Resource
* Dynamic Resource

**Waarom resources**Omdat veel stijlen en opmaak hetzelfde moeten worden gehouden zijn er variabelen nodig om het hierin op te slaan. Ook moet het meteen duidelijk zijn in de code welke opmaak er op is toegepast. Dit wordt gedaan met behulp van resources.   
  
Resources kunt u aanmaken in XAML of C#. Deze resources moeten dan eerst wel in een ResourceDictionary worden opgeslagen zodat het in blokken kan worden aangeroepen.  
  
**Noodzakelijk voor resources**Omdat elke resource moet worden geïdentificeerd moet deze een unieke key hebben. Dit wordt gedaan met static- of dynamic resource. Hierdoor weet het programma welke resource kan worden hergebruikt. Deze resource key geeft u als type mee in de resource tag.  
  
**Variabel**Het belangrijkste van een resource is het feit dat de resources variabel zijn. Hierbij kunt u eigenschappen aan resources toepassen en deze wijzigen. Wel moet er dan rekening gehouden worden met static resources.

## Static resource

Static resources worden voor het compileren uitgevoerd. Dit betekent dat static resources eenmaal wordt uitgevoerd. Tijdens het uitvoeren van de applicatie kunnen hierbij dan geen veranderingen worden aangepast. Static resources kunnen hierbij dan niet worden gerefereerd. De volgende XAML code laat zien hoe u een static resource aanroept:



**StaticResource:** *De resource type die u aangeeft voor het aanroepen van een bepaalde resource.*

**Key:** *De identiteit van de resource die u aanroept.*

**SNELLE OPZET**Een static resource zet u als volgt op:

1. Maak een nieuwe **.XAML** aan in **Visual Studio**
2. Maak een **ResourceDictionary tag** aan

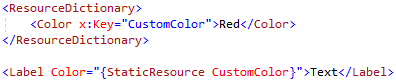


1. Maak in de **ResourceDictionary** een resource aan. Dit mag elke resource zijn die u wilt. In dit geval wordt er een **resource** **Color** aangemaakt met een **x:Key=”CustomColor”.**



1. Als laatst roept u de resource aan met **static resource**.

Eindresultaat

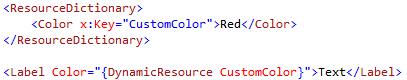


## Dynamic resource

Met dynamic resources kunt u een resource wijzigen tijdens het uitvoeren van de applicatie. Wat bij static resources het geval was dat je voor het compileren resources kunt uitvoeren en wijzigen is bij dynamic resources niet het geval.   
  
Deze kunt u dan alleen maar voor gebruik maken tijdens uitvoering. Het aanroepen van een dynamic resource lijkt heel veel als het aanroepen van de static resource. Hierbij is het enige verschil dat u in plaats van StaticResource een DynamicResource aanroept. De volgende opzet XAML code laat zien hoe.



**Refereer in C#**Het refereren met dynamic resource werkt wel en bij static resources niet omdat deze tijdens het uitvoeren van de applicatie kan worden gewijzigd. Hierbij wordt er van de volgende XAML code vanuit gegaan:



Om de CustomColor te refereren doet u als volgt:

  
Hiermee kunt u gemmakelijk de resources hergebruiken op elke gepaste object in code.

## Waarom static- of dynamic resource

Omdat een static resource alleen voor uitvoering van de applicatie wordt uitgevoerd is het vaak handig om met behulp van static resources een template te maken.   
  
Dit omdat er in templates niets veranderd hoeft te worden omdat deze resources niet dynamisch zijn. Het grote voordeel hierbij is dat de snelheid wordt behouden. Dit is niet het geval bij dynamic resources. Wel kunt u de applicatie updaten met behulp van dynamic resources. Hiermee wordt bedoeld dat u controls kunt maken die u wanneer u wilt kan aanpassen.  
  
Met static resources met behulp van ResourceDictionary kunt u verschillende thema’s maken. Als u dit met dynamic resources probeert kunnen de thema’s er heel anders uitzien omdat er tijdens uitvoering van de applicatie resources kunnen worden gewijzigd en niet statisch zijn.

Het beste is om altijd te beginnen met static resources. Alleen als u zeker weet dat u dynamic resources nodig heeft gebruikt u deze.   
  
De volgende voor- en nadelen nogmaals op een rijtje:  
  
**Static resource**

|  |  |
| --- | --- |
| + | - |
| Snelheid | Niet aan te passen tijdens uitvoering van de applicatie |
| Geschikt voor templates | Kan langer duren voordat de applicatie wordt opgestart |

**Dynamic resource**

|  |  |
| --- | --- |
| + | - |
| Updaten van de programma | Snelheid |
|  | Kan geen uitvoeringen doen voor het compileren |
|  | Niet geschikt voor templates |